МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №7_2

по дисциплине
«Структуры и алгоритмы обработки данных»

Тема. Графы: создание, алгоритмы обхода, важные задачи теории графов

Выполнил студент группы ИКБО-42-23 Голев С.С.

Принял ассистент Муравьёва Е.А.

Москва 2024

ОГЛАВЛЕНИЕ

ЦЕЛЬ РАБОТЫ

Получение практических навыков по выполнению операций над структурой данных граф.

УСЛОВИЯ ЗАДАЧ

Задание 1:

Вопросы.

Задание 2:

1. Разработать класс «Граф», обеспечивающий хранение и работу со структурой данных «граф», в соответствии с вариантом индивидуального задания:

- Реализовать метод ввода графа с клавиатуры, наполнение графа осуществлять с помощью метода добавления одного ребра.
- Реализовать методы, выполняющие задачи, определенные вариантом индивидуального задания.
- Разработать доступный способ (форму) вывода результирующего дерева на экран монитора.

2. Разработать программу, демонстрирующую работу всех методов класса.

3. Произвести тестирование программы на одном из графов, предложенных в таблице.

4. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Список смежных вершин	1. Составить программу нахождения кратчайших путей методом «Флойда». 2. Используя результат алгоритма вывести путь между вводимыми парами вершин.

1. ОТЧЁТ ПО ЗАДАНИЮ 1

1. Дайте определения понятиям: ориентированный граф, неориентированный граф, взвешенный граф, связный граф, центр графа, диаметр графа, матрица смежности.

Ориентированный граф — это граф, в котором каждое ребро имеет направление. Ребра представляют собой пары вершин (u, v), указывая, что существует связь от вершины u к вершине v, но не обязательно обратно.

Неориентированный граф — это граф, в котором ребра не имеют направления. Ребро между вершинами u и v указывает на взаимосвязь, где связь существует в обе стороны (u <-> v).

Взвешенный граф — это граф, в котором каждому ребру присвоено числовое значение (вес). Эти веса могут представлять расстояния, стоимости или другие количественные характеристики.

Связный граф — это граф, в котором существует путь между каждой парой вершин. В неориентированном графе это означает, что можно пройти от любой вершины к любой другой. В ориентированном графе каждая пара вершин должна быть связана по направлению.

Центр графа — это вершина (или несколько вершин), которая имеет минимальное расстояние до всех других вершин в графе. Центр позволяет оптимизировать расстояния до других узлов в графе.

Диаметр графа — это максимальное расстояние между любой парой вершин в графе. Он определяется как наибольшее количество ребер в кратчайшем пути между любыми двумя вершинами.

Матрица смежности — это квадратная матрица, которая используется для представления графа. Элементы матрицы указывают на наличие ребер между вершинами.

2. Что такое остовное дерево графа?

Это подмножество рёбер графа, которое соединяет все вершины графа без образования циклов и при этом имеет минимальное количество рёбер.

3. Какое количество ребер в остовном графе?

Если n- это количество вершин, то количество рёбер n-1.

4. Что такое кратчайший путь в графе?

Это путь между двумя вершинами, который имеет минимальную суммарную длину или вес среди всех возможных путей.

5. В чем отличие алгоритма Дейкстры от алгоритма Флойда-Уоршала. Какова вычислительная сложность каждого алгоритма по времени и памяти.

Применимость

Алгоритм Дейкстры:

Используется для нахождения кратчайшего пути от одной исходной вершины до всех остальных вершин в графе.

Подходит для графов с неотрицательными весами.

Алгоритм Флойда-Уоршелла:

Используется для нахождения кратчайших путей между всеми парами вершин в графе.

Может работать с графами, имеющими отрицательные веса, но без отрицательных циклов.

Подход

Алгоритм Дейкстры:

Работает жадным методом, выбирая в каждый момент времени вершину с минимальным расстоянием от начальной.

Алгоритм Флойда-Уоршелла:

Использует динамическое программирование, итеративно обновляя расстояния между всеми парами вершин.

Вычислительная сложность

Алгоритм Дейкстры:

По времени:

С использованием обычного массива: $O(V^2)$, где V — количество вершин.

С использованием очереди с приоритетом (например, бинарной

кучи): $O((V+E)logV)$, где E — количество рёбер.

По памяти: $O(V)$ (для хранения расстояний и вспомогательных данных).

Алгоритм Флойда-Уоршелла:

По времени: $O(V^3)$, так как требуется три вложенных цикла для перебора всех пар вершин.

По памяти: $O(V^2)$ (для хранения матрицы расстояний между всеми парами вершин).

2. ОТЧЁТ ПО ЗАДАНИЮ 2

2.1. Код используемый в программе

Опишем функции, используемые в программе для решения задач.

```
Graph(int vertices) {
        numVertices = vertices;

        graphList.resize(vertices + 1);
        dist.resize(vertices + 1, std::vector<int>(vertices + 1, std::numeric_limits<int>::max()));
        next.resize(vertices + 1, std::vector<int>(vertices + 1, std::numeric_limits<int>::min()));
        for (int i = 0; i <= numVertices; i++)
        {
            dist[i][i] = 0;
            next[i][i] = i;
        }
    }
```

Рисунок 2.1 – Конструктор графа

```
void addEdge(int src, int dest, int weight) {
        graphList[src].emplace_back(dest, weight);
        dist[src][dest] = weight;
        next[src][dest] = dest;
    }
```

Рисунок 2.2 – Функция добавления узла для ориентированного графа

```
void print_tree(int vertex, std::vector<bool>& visited, int depth = 0) {
        visited[vertex] = true;

        std::cout << std::setw(depth * 4) << " " << "+--[" << vertex  << ']' << std::endl;

        for (const Edge& edge : graphList[vertex]) {
            if (!visited[edge.vertex]) {
                std::cout << std::setw((depth + 1) * 4) << " " << "| " << edge.weight << std::endl;
                print_tree(edge.vertex, visited, depth + 1);
            }
        }
    }
void build_tree(int startVertex) {
        std::vector<bool> visited(numVertices + 1, false);
        print_tree(startVertex, visited);
    }
```

Рисунок 2.3 – Функции вывода графа в виде дерева

```
void floyd()
   {
      for (int k = 1; k <= numVertices; k++)
         for (int i = 1; i <= numVertices; i++)
            for (int j = 1; j <= numVertices; j++)
               if (dist[i][k]!= std::numeric_limits<int>::max() && dist[k][j] != std::numeric_limits<int>::max())
               {
                  if (dist[i][j] > dist[i][k] + dist[k][j])
                  {
                     dist[i][j] = dist[i][k] + dist[k][j];
                     next[i][j] = next[i][k];
                  }
               }
   }
```

Рисунок 2.4 – Реализация алгоритма Флойда

```
void find_path(int src, int dest)
   {
      if (next[src][dest] == std::numeric_limits<int>::min()) {
         std::cout << "Нет пути между вершинами " << src << " и " << dest << "." << std::endl;
         return;
      }

         std::cout << "Путь между вершинами " << src << " и " << dest << ": ";
         for (int at = src; at != dest; at = next[at][dest]) {
            std::cout << at << " ~ ";
         }
         std::cout << dest << std::endl;

   }
```

Рисунок 2.5 – Функция вывода кратчайшего пути в графе

2.3. Результаты тестирования

Протестируем функции, используемые во втором задании.

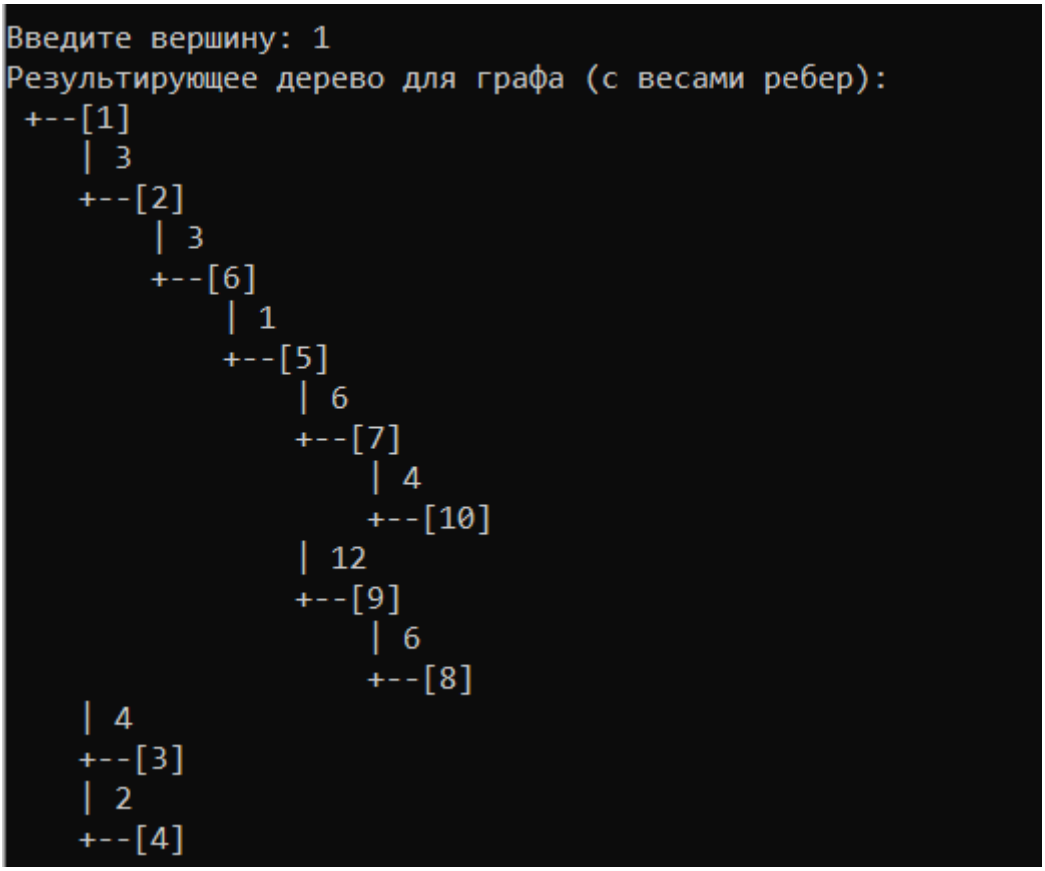

Рисунок 2.6 – Тестирование функции вывода

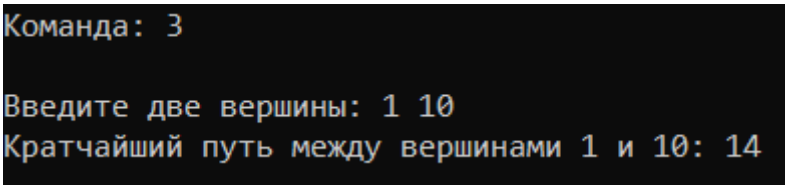

Рисунок 2.7 – Тестирование функции нахождения длинны пути

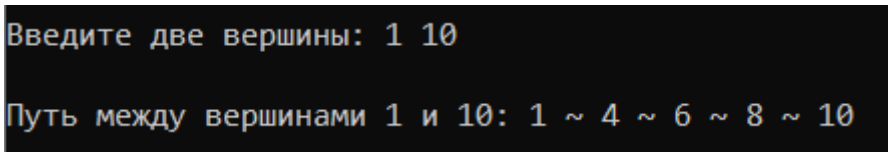

Рисунок 2.8 – Тестирование функции нахождения кратчайшего пути

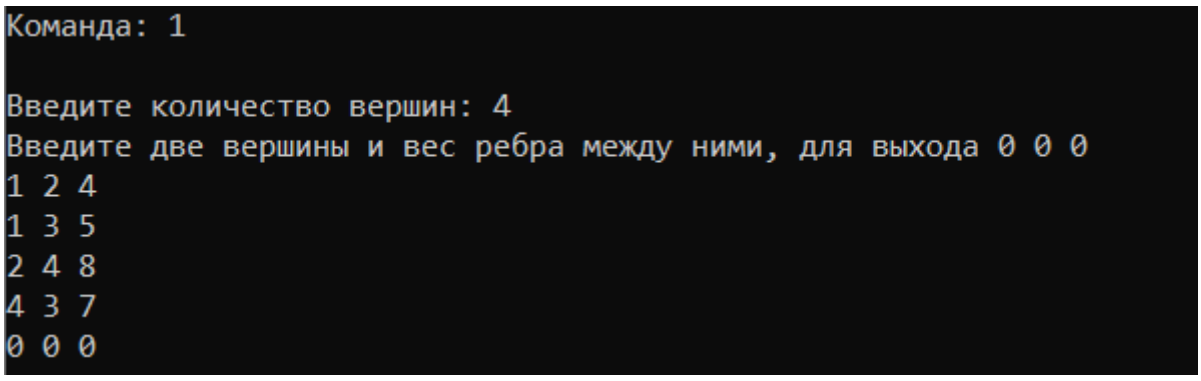

Рисунок 2.9 – Тестирование функции вставки (1/2 часть)

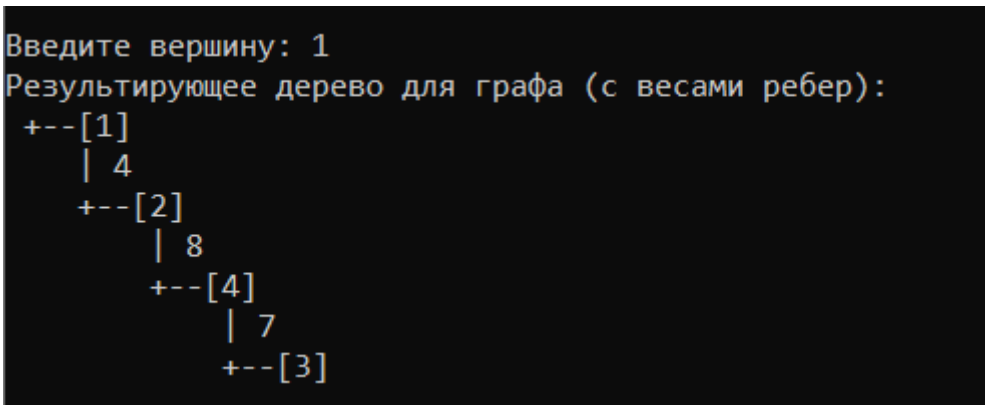

Рисунок 2.10 – Тестирование функции вставки (2/2 часть)

4.ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ

1. Лекции по Структуры и алгоритмы обработки данных / Рысин М. Л. Москва, МИРЭА — Российский технологический университет.

2. Материалы по дисциплине Структуры и алгоритмы обработки данных / Скворцова Л. А. Москва, МИРЭА — Российский технологический университет.